3. J2ME

Las características concretas de los teléfonos móviles ha obligado a los desarrolladores Java a construir un subconjunto del lenguaje y a reconfigurar sus principales bibliotecas para permitir su adaptación a un entorno con poca capacidad de memoria, poca velocidad de proceso y pantallas de reducidas dimensiones. Todo esto hace que necesitemos una plataforma nueva de desarrollo y ejecución sobre la que se centrará todo este capítulo: Java 2 Micro Edition, o de forma más conocida, J2ME.

3.1 Análisis comparativo

Antes de comenzar con los conceptos necesarios de Java Micro Edition vamos a realizar un estudio de las distintas versiones de Java existentes.

La empresa Sun MicroSystem, dispuesto a proporcionar las herramientas necesarias para cubrir las necesidades de todos los usuarios, creó distintas versiones de Java de acuerdo a las necesidades de cada uno. Según esto el paquete Java 2 lo podemos dividir en 3 ediciones distintas: J2SE (Java Standard Edition) orientada al desarrollo de aplicaciones independientes de la plataforma, J2EE (Java Enterprise Edition) orientada al entorno empresarial y J2ME (Java Micro Edition) orientada a dispositivos con capacidades restringidas. A continuación se analizarán cada una de ellas:

1. **Java 2 Platform, Standard Edition (J2SE)**: Se trata de la edición que posee la esencia del lenguaje Java. Tiene las siguientes características:

   - Inspirado en C++, pero con componentes de alto nivel.
   - Código independiente de la plataforma, ejecutado en el cliente a través de la máquina virtual JVM (Java Virtual Machine).
   - Modelo de seguridad tipo *sandbox*, proporcionado por la JVM, que permite controlar los accesos a un programa determinado.
   - Independencia del sistema operativo, mediante un juego completo de APIs de programación.

   Esta versión de Java contiene el conjunto básico de herramientas usadas para desarrollar Java Applets, así cómo las APIs orientadas a la programación de aplicaciones de usuario final.

2. **Java 2 Platform, Enterprise Edition (J2EE):** Esta versión está orientada al entorno empresarial, por lo que está pensado no para ser ejecutado en un equipo, sino para ejecutarse sobre una red de ordenadores de manera distribuida y remota mediante EJBs (Enterprise Java Beans). Esta edición está orientada especialmente al desarrollo de servicios web, servicios de nombres, persistencia de objetos, XML, autenticación, APIs para la gestión de transacciones, etc.

3. **Java 2 Platform, Micro Edition (J2ME)**: Esta versión de Java está enfocada a la aplicación de la tecnología Java en dispositivos electrónicos con capacidades computacionales y gráficas muy reducidas, tales como teléfonos móviles, PDAs o electrodomésticos inteligentes. Esta edición tiene unos componentes básicos que la diferencian de las otras versiones, como el uso de una máquina virtual denominada KVM (Kilo Virtual Machine, debido a que requiere sólo unos pocos Kilobytes de memoria para funcionar) en vez del uso de la JVM clásica, y otras diferencias que ya iremos viendo más adelante. La siguiente figura nos muestra la arquitectura del lenguaje Java:

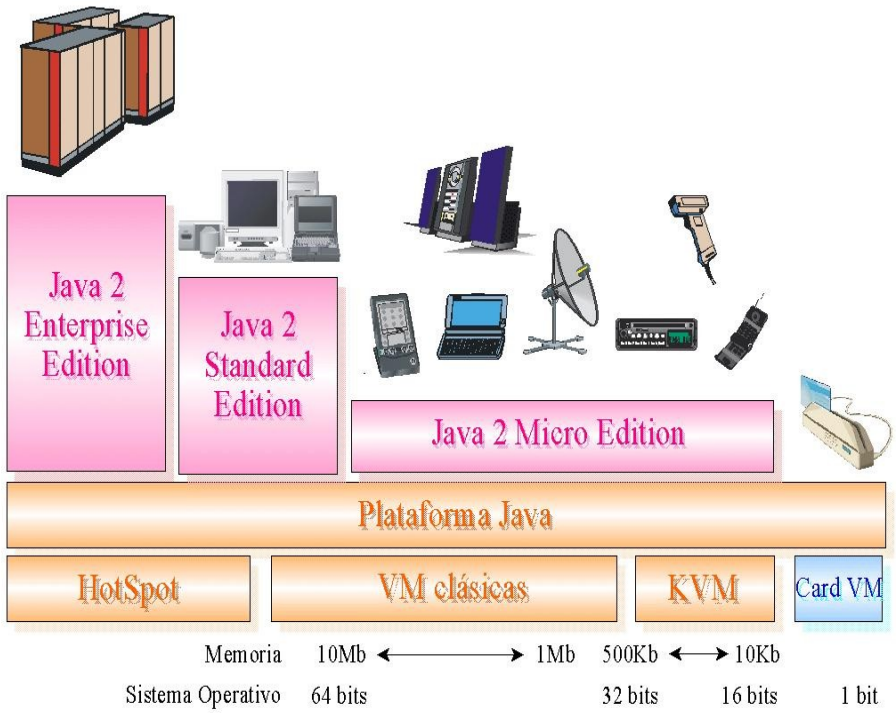

Figura 3.1. Arquitectura de la plataforma Java 2 de Sun.

En la actualidad no es realista ver Java como un simple lenguaje de programación, si no como un conjunto de tecnologías que abarca a todos los ámbitos de la computación con dos elementos en común:

- El código fuente en lenguaje Java es compilado a código intermedio interpretado por una Java Virtual Machine (JVM), por lo que el código ya compilado es independiente de la plataforma.

- Todas las tecnologías comparten un conjunto más o menos amplio de APIs básicas del lenguaje, agrupadas principalmente en los paquetes java.lang y java.io.

Un claro ejemplo de éste último punto es que J2ME contiene una mínima parte de las APIs de Java. Esto es debido a que la edición estándar de APIs de Java ocupa 20 Mb, y los dispositivos pequeños disponen de una cantidad de memoria mucho más reducida. En concreto, J2ME usa 37 clases de la plataforma J2SE provenientes de los paquetes java.lang, java.io, java.util. Esta parte de la API que se mantiene fija forma parte de lo que se denomina "configuración" y se explicará más detenidamente posteriormente. Otras diferencias con la plataforma J2SE vienen dadas por el uso de

una máquina virtual distinta de la clásica JVM denominada KVM. Esta KVM tiene unas restricciones que hacen que no posea todas las capacidades incluidas en la JVM.

Actualmente, J2EE es un superconjunto de J2SE pues contiene toda la funcionalidad de éste y más características, así como J2ME es un subconjunto de J2SE (excepto por el paquete javax.microedition) ya que, como se ha mencionado, contiene varias limitaciones con respecto a J2SE.

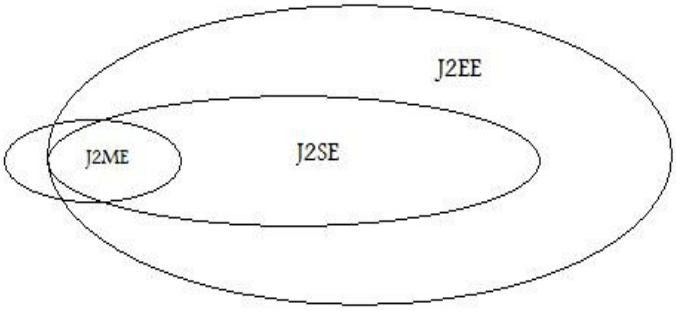

Figura 3.2 Relación entre las APIs de la plataforma Java.

3.2 Nociones básicas de J2ME

En este apartado vamos a analizar los distintos componentes que forman parte de la tecnología J2ME.

- Máquinas virtuales Java con diferentes requisitos, cada una para los distintos tipos de pequeños dispositivos.

- Configuraciones, que son un conjunto de clases básicas orientadas a conformar el corazón de las implementaciones para dispositivos de características específicas. Existen 2 configuraciones definidas en J2ME: Connected Limited Device Configuration (CLDC) enfocada a dispositivos con restricciones de procesamiento y memoria, y Connected Device Configuration (CDC) enfocada a dispositivos con más recursos.

- Perfiles, que son unas bibliotecas Java de clases específicas orientadas a implementar funcionalidades de más alto nivel para familias específicas de dispositivos.

Un entorno de ejecución determinado de J2ME se compone entonces de una selección de:

- Máquina virtual.

- Configuración.

- Perfil.
- Paquetes Opcionales.

La arquitectura de un entorno de ejecución la podemos ver en la Figura 1.3. A continuación estudiaremos en profundidad cada uno de estos tres componentes.

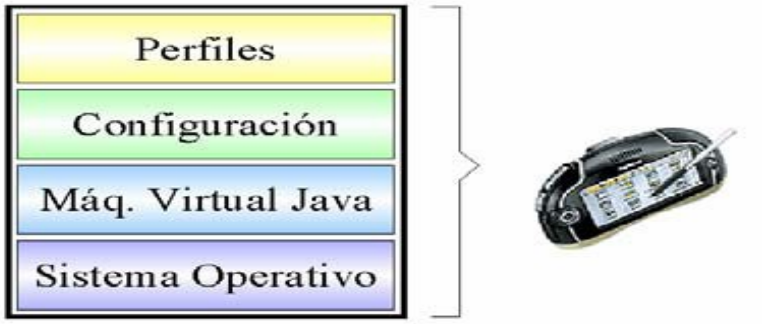

Figura 3.3. Entorno de ejecución

3.2.1 Máquinas virtuales J2ME

Una máquina virtual de Java (JVM) es un programa encargado de interpretar código intermedio (bytecode) de los programas Java precompilados a código máquina ejecutable por la plataforma, efectuar las llamadas pertinentes al sistema operativo subyacente y observar las reglas de seguridad y corrección de código definidas para el lenguaje Java. De esta forma, la JVM proporciona al programa Java independencia de la plataforma con respecto al hardware y al sistema operativo subyacente.

Las implementaciones tradicionales de JVM son, en general, muy pesadas en cuanto a memoria ocupada y requerimientos computacionales. J2ME define varias JVMs de referencia adecuadas al ámbito de los dispositivos electrónicos que, en algunos casos, suprimen algunas características con el fin de obtener una implementación menos exigente.

Ya hemos visto que existen 2 configuraciones CLDC y CDC, cada una con unas características propias que veremos en profundidad más adelante. Como consecuencia, se requieren máquinas virtuales adaptadas a cada una de esas configuraciones. La VM (Virtual Machine) de la configuración CLDC se denomina KVM y la de la configuración CDC se denomina CVM. Veremos a continuación las características principales de cada una de ellas:

- KVM

Se corresponde con la Máquina Virtual más pequeña desarrollada por Sun. Su nombre KVM proviene de Kilobyte (haciendo referencia a la baja ocupación de memoria, entre 40Kb y 80Kb). Se trata de una implementación de Máquina Virtual reducida y especialmente orientada a dispositivos con bajas capacidades computacionales y de memoria. Es modulable, completa y rápida, sin sacrificar características para las que fue diseñada.

Sin embargo, presenta serias limitaciones, ya que no incluye ciertas características propias de la clásica máquina virtual Java JVM.

- CVM

La CVM (Compact Virtual Machine) ha sido tomada como Máquina Virtual Java de referencia para la configuración CDC y soporta las mismas características que la Máquina Virtual de J2SE. Está orientada a dispositivos electrónicos con procesadores de 32 bits de gama alta y en torno a 2Mb o más de memoria RAM. Las características que presenta esta Máquina Virtual son, entre otras, sistema de memoria avanzado, portabilidad, rápida sincronización, baja ocupación en memoria de clases, etc.

3.2.2 Configuraciones

Una configuración es el conjunto mínimo de APIs Java que permiten desarrollar aplicaciones para un grupo de dispositivos. Éstas APIs describen las características básicas, comunes a todos los dispositivos:

- Características soportadas del lenguaje de programación Java.
- Características soportadas por la Máquina Virtual Java.
- Bibliotecas básicas de Java y APIs soportadas.

Como ya hemos visto con anterioridad, existen dos configuraciones en J2ME: CLDC, orientada a dispositivos con limitaciones computacionales y de memoria y CDC, orientada a dispositivos con menos limitaciones. A continuación, se analizarán cada una de estas configuraciones con más profundidad.

- Configuración de dispositivos con conexión, CDC (Connected Device Configuration)

La CDC está orientada a dispositivos con cierta capacidad computacional y de memoria. Por ejemplo, decodificadores de televisión digital, televisores con Internet, algunos electrodomésticos y sistemas de navegación en automóviles. CDC usa una Máquina Virtual Java similar en sus características a una de J2SE, pero con limitaciones en el apartado gráfico y de memoria del dispositivo. Ésta Máquina Virtual es la que hemos visto como CVM (Compact Virtual Machine).

Esta configuración CDC está enfocada a dispositivos con las siguientes capacidades:

a) Procesador de 32 bits.
b) Disponer de 2 Mb o más de memoria total, incluyendo memoria RAM y ROM.
c) Poseer la funcionalidad completa de la Máquina Virtual Java2.
d) Conectividad a algún tipo de red.

La CDC está basada en J2SE v1.3 e incluye varios paquetes Java de la edición estándar. Las peculiaridades de la CDC están contenidas principalmente en el paquete javax.microedition.io, que incluye soporte para comunicaciones Http y basadas en datagramas. La Tabla 3.1 nos muestra las librerías incluidas en la CDC.

Nombre de Paquete CDC	Descripción
java.io	Clases e interfaces estándar de E/S.
java.lang	Clases básicas del lenguaje.
java.lang.ref	Clases de referencia.
java.lang.reflect	Clases e interfaces de reflection.
java.math	Paquete de matemáticas.
java.net	Clases e interfaces de red.
java.security	Clases e interfaces de seguridad
java.security.cert	Clases de certificados de seguridad.
java.text	Paquete de texto.
java.util	Clases de utilidades estándar.
java.util.jar	Clases y utilidades para archivos JAR.
java.util.zip	Clases y utilidades para archivos ZIP y comprimidos.
javax.microedition.io	Clases e interfaces para conexión genérica CDC.

Tabla 3.1. Librerías de configuración CDC

- Configuración de dispositivos limitados con conexión, CLDC (Connected Limited Device Configuration)

La CLDC está orientada a dispositivos dotados de conexión y con limitaciones en cuanto a capacidad gráfica, cómputo y memoria. Un ejemplo de éstos dispositivos son: teléfonos móviles, buscapersonas (pagers), PDAs, organizadores personales, etc. CDLC está orientada a dispositivos con ciertas restricciones. Algunas de éstas restricciones vienen dadas por el uso de la KVM, necesaria al trabajar con la CLDC debido a su pequeño tamaño. Los dispositivos que usan CLDC deben cumplir los siguientes requisitos:

a) Procesador de 16 o 32 bits con al menos 25 Mhz de velocidad.

b) Disponer entre 160 Kb y 512 Kb de memoria total disponible. Como mínimo se debe disponer de 128 Kb de memoria no volátil para la Máquina Virtual Java y las bibliotecas CLDC, y 32 Kb de memoria volátil para la Máquina Virtual en tiempo de ejecución.

c) Ofrecer bajo consumo, debido a que éstos dispositivos trabajan con suministro de energía limitado, normalmente baterías.

d) Tener conexión a algún tipo de red, normalmente sin cable, con conexión intermitente y ancho de banda limitado (unos 9600 bps).

La configuración CLDC aporta las siguientes funcionalidades a los dispositivos:

a) Un subconjunto del lenguaje Java y todas las restricciones de su Máquina Virtual (KVM).

b) Un subconjunto de las bibliotecas Java del núcleo.

c) Soporte para E/S básica.

d) Soporte para acceso a redes.

e) Seguridad.

La Tabla 3.2 nos muestra las librerías incluidas en la CLDC.

Nombre de paquete CLDC	Descripción
java.io	Clases y paquetes estándar de E/S. Subconjunto de J2SE.
java.lang	Clases e interfaces de la Máquina Virtual. Subconj. de J2SE.
java.util	Clases, interfaces y utilidades estándar. Subconj. de J2SE.
javax.microedition.io	Clases e interfaces de conexión genérica CLDC

Tabla 3.2 Librerías de configuración CLDC

3.2.3 Perfiles

Un perfil es un conjunto de APIs orientado a un ámbito de aplicación determinado. Los perfiles identifican un grupo de dispositivos por la funcionalidad que proporcionan (electrodomésticos, teléfonos móviles, etc.) y el tipo de aplicaciones que se ejecutarán en ellos. Las librerías de la interfaz gráfica son un componente muy importante en la definición de un perfil.

El perfil establece unas APIs que definen las características de un dispositivo, mientras que la configuración hace lo propio con una familia de ellos. Esto hace que a la hora de construir una aplicación se cuente tanto con las APIs del perfil como de la configuración. Tenemos que tener en cuenta que un perfil siempre se construye sobre una configuración determinada. De este modo, podemos pensar en un perfil como un conjunto de APIs que dotan a una configuración de funcionalidad específica.

Anteriormente vimos que para una configuración determinada se usaba una máquina virtual Java específica. Con los perfiles ocurre lo mismo. Existen perfiles que se construirán sobre la configuración CDC y otros sobre la configuración CLDC. Para la configuración CDC existen los siguientes perfiles:

- Foundation Profile
- Personal Profile
- RMI Profile

y para la configuración CDLC existen los siguientes:

- PDA Profile

- Mobile Information Device Profile (MIDP)

En la siguiente figura se podría ver cómo quedaría el entorno de ejecución al completo.

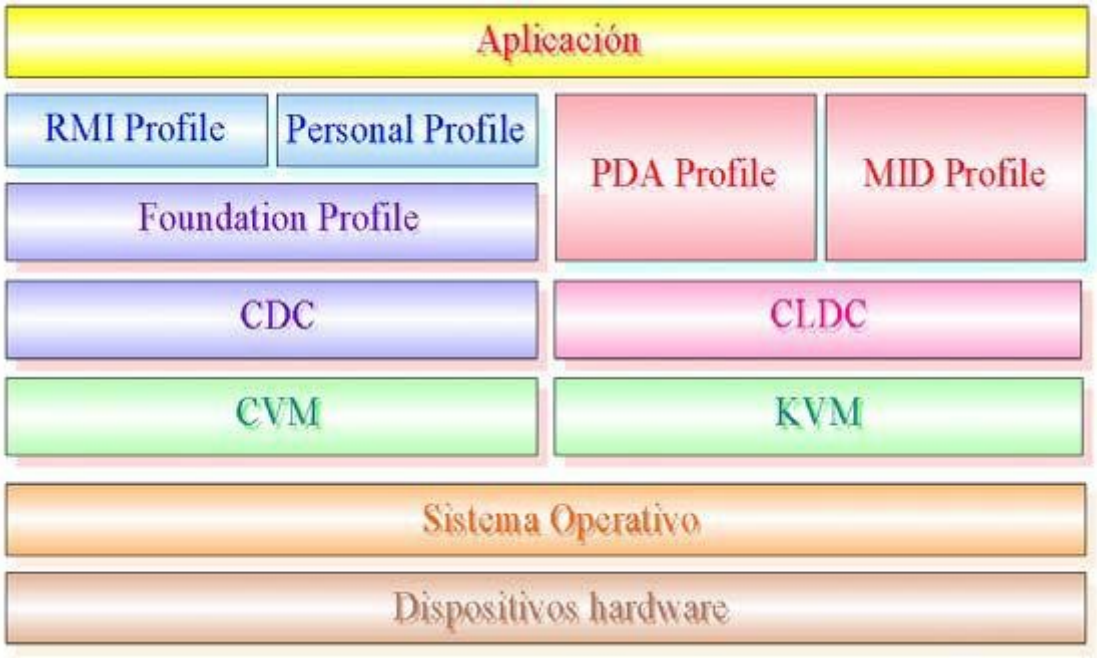

Figura 3.4 Arquitectura del entorno de ejecución J2ME

Un perfil puede ser construido sobre cualquier otro. Sin embargo, una plataforma J2ME sólo puede contener una configuración. A continuación vamos a ver con detenimiento cada uno de estos perfiles:

- **Foundation Profile:** Este perfil define una serie de APIs sobre la CDC orientadas a dispositivos que carecen de interfaz gráfica como, por ejemplo, decodificadores de televisión digital. Este perfil incluye gran parte de los paquetes de la J2SE, pero excluye totalmente los paquetes "java.awt" *Abstract Windows Toolkit* (AWT) y "java.swing" que conforman la interfaz gráfica de usuario (GUI) de J2SE. Si una aplicación requiriera una GUI, entonces sería necesario un perfil adicional.

- **Personal Profile:** El *Personal Profile* es un subconjunto de la plataforma J2SE v1.3, y proporciona un entorno con un completo soporte gráfico AWT. El objetivo es el de dotar a la configuración CDC de una interfaz gráfica completa, con capacidades web y soporte de *applets* Java. Este perfil requiere una implementación del *Foundation Profile*.

- **RMI Profile:** Este perfil requiere una implementación del *Foundation Profile,* puesto que se construye encima de él. El perfil RMI soporta un subconjunto de las APIs J2SE v1.3 RMI. Algunas características de estas APIs se han eliminado del perfil RMI debido a las limitaciones de cómputo y memoria de los dispositivos.

- **PDA Profile:** El PDA Profile está construido sobre CLDC. Pretende abarcar PDAs de gama baja, tipo Palm, con una pantalla y algún tipo de puntero (ratón o lápiz) y una resolución de al menos 20000 pixels (al menos 200x100 pixels) con un factor 2:1.

- **Mobile Informa ion Device Profile (MIDP):** Este perfil está construido sobre la configuración CLDC. Al igual que CDLC fue la primera configuración definida para J2ME, MIDP fue el primer perfil definido para esta plataforma. Este perfil está orientado para dispositivos con las siguientes características:

    a) Reducida capacidad computacional y de memoria.
    b) Conectividad limitada (en torno a 9600 bps).
    c) Capacidad gráfica muy reducida (mínimo un display de 96x54 pixels
    d) monocromo).
    e) Entrada de datos alfanumérica reducida.
    f) 128 Kb de memoria no volátil para componentes MIDP.
    g) 8 Kb de memoria no volátil para datos persistentes de aplicaciones.
    h) 32 Kb de memoria volátil en tiempo de ejecución para la pila Java.

El perfil MIDP establece las capacidades del dispositivo, por lo tanto, especifica las APIs relacionadas con:

a) La aplicación (semántica y control de la aplicación MIDP).
b) Interfaz de usuario.
c) Almacenamiento persistente.
d) Trabajo en red.
e) Temporizadores.

En la Tabla 3.3 podemos ver cuáles son los paquetes que están incluidos en el perfil MIDP.

Paquetes del MIDP	Descripción
javax.microedition.lcdui	Clases e interfaces para GUIs
javax.microedition.rms	*Record Management Storage*. Soporte para el almacenamiento persistente del dispositivo
javax.microedition.midlet	Clases de definición de la aplicación
javax.microedition.io	Clases e interfaces de conexión genérica
java.io	Clases e interfaces de E/S básica
java.lang	Clases e interfaces de la Máquina Virtual
java.util	Clases e interfaces de utilidades estándar

Tabla 3.3 Librerías del perfil MIDP

Las aplicaciones que se realizan utilizando MIDP reciben el nombre de *MIDlets*. **Decimos así que un MIDlet es una aplicación Java realizada con el perfil MIDP sobre la configuración CLDC**. Desde un punto de vista práctico MIDP es el único perfil actualmente disponible.

3.3 Configuración CDLC

En el siguiente apartado se va a analizar con más profundidad la configuración CDLC, los aspectos en los que se centra y las características de cada uno de ellos.

La configuración CLDC se ocupa de las siguientes áreas:

- Lenguaje Java y características de la máquina virtual.
- Librerías del núcleo de Java (java.lang.* y java.util.*).
- Entrada / Salida.
- Comunicaciones.
- Seguridad.
- Internacionalización.

Una configuración de J2ME especifica un subconjunto de características soportadas por el lenguaje de programación Java, un subconjunto de funciones de la configuración para la máquina virtual de Java, el trabajo en red, seguridad, instalación y, posiblemente, otras APIs de programación, todo lo necesario para soportar un cierto tipo de productos.

CLDC es la base para uno o más perfiles. Un perfil de J2ME define un conjunto adicional de APIs y características para un mercado concreto, dispositivo determinado o industria. Las configuraciones y los perfiles están mas definidos exactamente en la publicación *Configurations and Profiles Architecture Specification, Java 2 Platform Micro Edition (J2ME), Sun Microsystems, Inc*.

3.3.1 Introducción

El objetivo principal de la especificación CLDC es definir un estándar, para pequeños dispositivos conectados con recursos limitados con las siguientes características:

- 160 Kb a 512 Kb de memoria total disponible para la plataforma Java.
- Procesador de 16 bits o 32 bits.
- Bajo consumo, normalmente el dispositivo usa una batería.
- Conectividad a algún tipo de red, normalmente inalámbrica, con conexión intermitente y ancho de banda limitado (unos 9600 bps).

Teléfonos móviles, PDAs y terminales de venta, son algunos de los dispositivos que podrían ser soportados por esta especificación. Esta configuración J2ME define los componentes mínimos y librerías requeridas por dispositivos conectados pequeños.

3.3.2 Objetivos

Aunque el principal objetivo de la especificación CLDC es definir un estándar para dispositivos pequeños y conectados con recursos limitados, existen otros objetivos que son los siguientes:

- Extensibilidad: Uno de los grandes beneficios de la tecnología Java en los pequeños dispositivos es la distribución dinámica y de forma segura de contenido interactivo y aplicaciones sobre diferentes tipos de red. Hace unos años, estos dispositivos se fabricaban con unas características fuertemente definidas y sin capacidad apenas de extensibilidad. Los desarrolladores de dispositivos empezaron a buscar soluciones que permitieran construir dispositivos extensibles que soportaran una gran variedad de aplicaciones provenientes de

terceras partes. Uno de los principales objetivos de CDLC es permitir esta extensibilidad mediante el uso del lenguaje Java para la distribución de contenido para estos dispositivos.

- Desarrollo de aplicaciones por terceras partes: La especificación CLDC solo deberá incluir librerías de alto nivel que proporcionen suficiente capacidad de programación para desarrollar aplicaciones por terceras partes. Por esta razón, las APIs de red incluidas en CLDC deberían proporcionar al programador una abstracción de alto nivel como, por ejemplo, la capacidad de transferir archivos enteros, aplicaciones o páginas web, en vez de requerir que el programador conozca los detalles de los protocolos de transmisión para una red específica.

3.3.3 Requerimientos

Podemos clasificar los requerimientos de la configuración CDLC en requerimientos *hardware, software* y requerimientos basados en las características Java de la plataforma J2ME.

- Requerimientos hardware: CLDC está diseñado para ejecutarse en una gran variedad de dispositivos, desde aparatos de comunicación inalámbricos como teléfonos móviles, buscapersonas, hasta organizadores personales, terminales de venta, etc. Las capacidades del hardware de estos dispositivos varían considerablemente y por esta razón, los únicos requerimientos que impone la configuración CLDC son los de memoria. La configuración CLDC asume que la máquina virtual, librerías de configuración, librerías de perfil y la aplicación debe ocupar una memoria entre 160 Kb y 512 Kb.

- Requerimientos software: Al igual que las capacidades hardware, el software incluido en los dispositivos CLDC varía considerablemente. Por ejemplo, algunos dispositivos pueden poseer un sistema operativo (S.O.) completo que soporta múltiples procesos ejecutándose a la vez y con sistema de archivos jerárquico. Otros muchos dispositivos pueden poseer un software muy limitado sin noción alguna de lo que es un sistema de ficheros. Dentro de esta variedad, CLDC define unas mínimas características que deben poseer el software de los dispositivos CLDC. Generalmente, la configuración CLDC asume que el dispositivo contiene un mínimo Sistema Operativo encargado del manejo del hardware de éste. Este S.O. debe proporcionar al menos una entidad de planificación para ejecutar la JVM.

- Requerimientos J2ME: CLDC es definida como la configuración de J2ME. Esto tiene importantes implicaciones para la especificación CLDC:

    - Una configuración J2ME solo deber definir un complemento mínimo de la tecnología Java. Todas las características incluidas en una configuración deben ser generalmente aplicables a una gran variedad de dispositivos. Características específicas para un dispositivo, mercado o industria deben ser definidas en un perfil en vez de en el CLDC. Esto significa que el alcance de CLDC está limitado y debe ser complementado generalmente por perfiles.

    - El objetivo de la configuración es garantizar portabilidad e interoperabilidad entre varios tipos de dispositivos con recursos limitados. Una configuración no debe definir ninguna característica opcional. Esta limitación tiene un impacto significativo en lo que se puede incluir en una configuración y lo que no. La funcionalidad más específica debe ser

definida en perfiles

3.3.4 Seguridad

Debido a las características de los dispositivos englobados bajo CLDC, en los que se hace necesaria la descarga de aplicaciones y la ejecución de éstas en dispositivos que almacenan información muy personal, es imprescindible la seguridad en estos dispositivos. Hay que asegurar la integridad de los datos transmitidos y de las aplicaciones. Éste modelo de seguridad no es nuevo, ya que la ejecución de applets (programas Java que se ejecutan en un navegador web) se realiza en una zona de seguridad denominada *sandbox*. Los dispositivos englobados en CLDC se encuentran ante un modelo similar al de los applets.

Este modelo establece que sólo se pueden ejecutar algunas acciones que se consideran seguras, por lo que existen algunas funcionalidades críticas que están fuera del alcance de las aplicaciones. De esta forma, las aplicaciones ejecutadas en estos dispositivos deben cumplir unas condiciones previas:

- Los ficheros de clases Java deben ser verificados como aplicaciones Java válidas.
- Sólo se permite el uso de APIs autorizadas por CLDC.
- No está permitido cargar clases definidas por el usuario.
- Sólo se puede acceder a características nativas que entren dentro del CLDC.
- Una aplicación ejecutada bajo KVM no debe ser capaz de dañar el dispositivo dónde se encuentra. De esto se encarga el verificador de clases que se asegura que no haya referencias a posiciones no válidas de memoria. También comprueba que las clases cargadas no se ejecuten de una manera no permitida por las especificaciones de la Máquina Virtual.

3.3.5 Librerías CDLC

Las versiones de Java J2EE y J2SE proporcionan un gran conjunto de librerías para el desarrollo de aplicaciones empresariales en servidores y ordenadores de sobremesa respectivamente. Desafortunadamente, estas librerías requieren gran capacidad para ser ejecutadas y es no es posible el almacenamiento de todas estas librerías en pequeños dispositivos con recursos limitados.

Por este motivo, el objetivo general para el diseño de librerías Java para CLDC es proporcionar un conjunto mínimo de librerías útiles para el desarrollo de aplicaciones y definición de perfiles para una variedad de pequeños dispositivos. Dadas las estrictas restricciones de memoria y diferencias en las características de estos dispositivos, es completamente imposible ofrecer un conjunto de librerías que satisfagan a todo el mundo.

Las librerías CLDC pueden ser divididas en dos categorías:

- Clases que son un subconjunto de las librerías de J2SE.

  CLDC proporciona un conjunto de clases heredadas de la plataforma J2SE. En total, usa unas 37 clases provenientes de los paquetes java.lang, java.util y java.io. Cada una de

estas clases debe ser idéntica o ser un subconjunto de la correspondiente clase de J2SE. Tanto los métodos como la semántica de cada clase deben permanecer invariables. En las tablas siguientes se muestran estas clases agrupadas según su funcionalidad.

Clases de Datos (Subconjunto de java.lang)
java.lang.Boolean
java.lang.Byte
java.lang.Character
java.lang.Integer
java.lang.Long
java.lang.Short

Clases de sistema (Subconjunto de java.lang)
java.lang.Class
java.lang.Object
java.lang.Runnable
java.lang.Runtime
java.lang.String
java.lang.Stringbuffer
java.lang.System
java.lang.Thread
java.lang.Throwable

Tabla 3.4 Clases de datos y sistema heredadas de J2SE.

Clases de E/S (Subconjunto de java.io)
java.io.ByteArrayInputStream
java.io.ByteArrayOutputStream
java.io.DataInput
java.io.DataOutput
java.io.DataInputStream
java.io.DataOutputStream
java.io.InputStream
java.io.InputStreamReader
java.io.OutputStream
java.io.OutputStreamWriter
java.io.PrintStream
java.io.Reader
java.io.Writer

Clases de Utilidades (Subconjunto de java.util)
java.util.Calendar
java.util.Date
java.util.Enumeration
java.util.Hashtable
java.util.Random
java.util.Stack
java.util.TimeZone
java.util.Vector

Tabla 3.5 Clases de E/S y de utilidades heredadas de J2SE

- Clases específicas de CLDC.

  La plataforma J2SE contiene los paquetes java.io y java.net encargados de las operaciones de E/S. Debido a las limitaciones de memoria de CLDC no es posible incluir dentro de él a todas las clases de estos paquetes. Ya hemos visto que CLDC hereda algunas clases del paquete java.io, pero no hereda ninguna clase relacionada con la E/S de ficheros, por ejemplo. Esto es debido a la gran variedad de dispositivos que abarca CLDC, ya que, para éstos puede resultar innecesario manejar ficheros. Para suplir estas "carencias" CLDC posee

un conjunto de clases más genérico para la E/S y la conexión en red que recibe el nombre de "*Generic Connection Framework*". Estas clases están incluidas en el paquete javax.microedition.io y se muestran en la tabla siguiente.

Clase	Descripción
Connector	Clase genérica que puede crear cualquier tipo de conexión.
Connection	Interfaz que define el tipo de conexión más genérica.
InputConnection	Interfaz que define una conexión de *streams* de entrada.
OutputConnection	Interfaz que define una conexión de *streams* de salida.
StreamConnection	Interfaz que define una conexión basada en *streams*.
ContentConnection	Extensión a StreamConnection para trabajar con datos.
Datagram	Interfaz genérico de datagramas.
DatagramConnection	Interfaz que define una conexión basada en datagramas.
StreamConnectionNotifier	Interfaz que notifica una conexión. Permite crear una conexión en el lado del servidor.

Tabla 3.6 Clases e interfaces incluidos en el paquete javax.microedition.io

3.4 Los MIDlets

3.4.1 Introducción

Los *MIDlets* son aplicaciones creadas usando la especificación MIDP. Están diseñados para ser ejecutados, en dispositivos con poca capacidad gráfica, de cómputo y de memoria. En estos dispositivos no disponemos de líneas de comandos donde poder ejecutar las aplicaciones que queramos, sino que reside en él un *software* que es el encargado de ejecutar los *MIDlets* y gestionar los recursos que éstos ocupan. En el siguiente apartado se va a analizar dicho s*oftware,* el gestor de aplicaciones.

El gestor de aplicaciones o AMS (*Application Management System*) es el *software* encargado de gestionar los *MIDlets*. Este *software* reside en el dispositivo y es el que nos permite ejecutar, pausar o destruir nuestras aplicaciones J2ME. A partir de ahora nos referiremos a él con las siglas de sus iniciales en inglés AMS. El AMS realiza dos grandes funciones:

- Por un lado gestiona el ciclo de vida de los *MIDlets*.
- Por otro, es el encargado de controlar los estados por los que pasa el *MIDlet* mientras está en la memoria del dispositivo, es decir, en ejecución.

3.4.2 Ciclo de vida de un MIDlet

El ciclo de vida de un MIDlet pasa por 5 fases: descubrimiento, instalación, ejecución, actualización y borrado, tal y como se muestra en la siguiente figura:

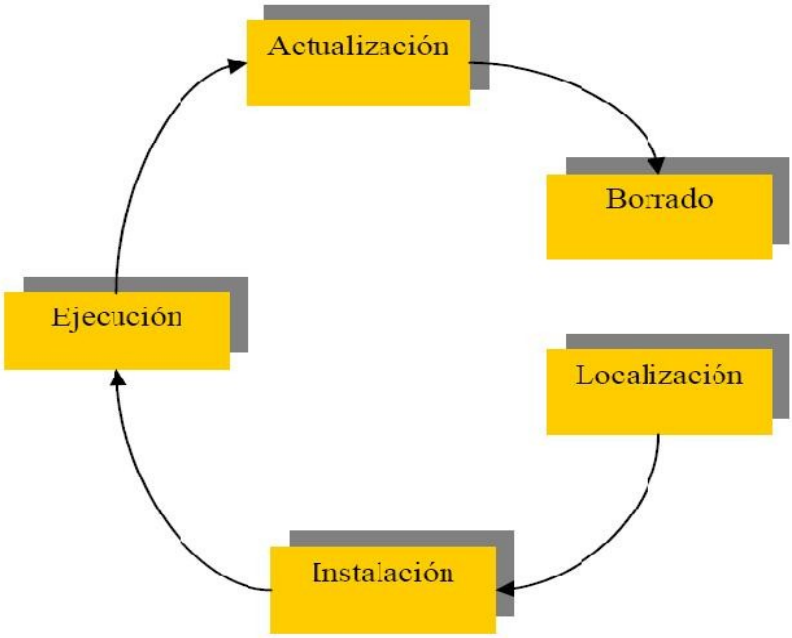

Figura 3.5 Ciclo de vida de un MIDlet

El AMS es el encargado de gestionar cada una de estas fases de la siguiente manera:

- Descubrimiento: Esta fase es la etapa previa a la instalación del MIDlet y a través de ella se selecciona la aplicación a descargar. Por tanto, el gestor de aplicaciones debe proporcionar los mecanismos necesarios para realizar la elección del MIDlet a descargar. El AMS puede ser capaz de realizar la descarga de aplicaciones de diferentes maneras, dependiendo de las capacidades del dispositivo. Por ejemplo, esta descarga la podemos realizar mediante un cable conectado a un ordenador o mediante una conexión inalámbrica.

- Instalación: Una vez descargado el MIDlet en el dispositivo, comienza el proceso de instalación. En esta fase el gestor de aplicaciones controla todo el proceso informando al usuario tanto de la evolución de la instalación como de si existiese algún problema durante ésta. Cuándo un MIDlet está instalado en el dispositivo, todas sus clases, archivos y almacenamiento persistente están preparados y listos para su uso.

- Ejecución: Mediante el gestor de aplicaciones es posible iniciar la ejecución de los *MIDlets*. En esta fase, el AMS tiene la función de gestionar los estados del MIDlet en función de los eventos que se produzcan durante esta ejecución.

- Actualización: El AMS tiene que ser capaz de detectar después de una descarga si el MIDlet descargado es una actualización de un MIDlet ya presente en el dispositivo. Si es así, nos tiene que informar de ello, además de darnos la oportunidad de decidir si queremos realizar la actualización pertinente o no.

- Borrado: En esta fase el AMS es el encargado de borrar el MIDlet seleccionado del dispositivo. El AMS nos pedirá confirmación antes de proceder a su borrado y nos informará de cualquier circunstancia que se produzca.

3.4.3 Estados de un MIDlet

Durante su ejecución, un MIDlet pasa por 3 estados diferentes. Como ya hemos visto en el apartado anterior, estos tres estados son:

- Activo: El MIDlet está actualmente en ejecución.
- Pausa: El *MIDlet* no está actualmente en ejecución. En este estado el *MIDlet* no debe usar ningún recurso compartido. Para volver a pasar a ejecución tiene que cambiar su estado a Activo.
- Destruido: El *MIDlet* no está en ejecución ni puede transitar a otro estado. Además se liberan todos los recursos ocupados por el *MIDlet*.

3.4.4 Librerías MIDlet

A continuación. Estudiaremos las librerías más significativas de un MIDlet, así como sus diferentes clases y métodos.

El paquete javax.microedition.midlet define las aplicaciones MIDP y su comportamiento con respecto al entorno de ejecución. Como ya sabemos, una aplicación creada usando MIDP es un *MIDlet*. En la Tabla 3.1 podemos ver cuáles son las clases que están incluidas en este paquete:

Clases	Descripción
MIDlet	Aplicación MIDP.
*MIDlets*tateChangeException	Indica que el cambio de estado ha fallado.

Tabla 3.7 Clases del paquete javax.microedition.midlet

A continuación nos centraremos en el paquete javax.microedition.lcdui (Interfaz de usuario con pantalla LCD), que es donde se encuentran los elementos gráficos a usar.

Teniendo en cuenta la diversidad de aplicaciones que podemos realizar para los dispositivos MID (llamaremos así a los dispositivos que soportan MIDP), y los elementos que nos proporcionan la configuración CLDC y el perfil MIDP, vamos a dividir a estos elementos en dos grupos:

1. Por un lado estudiaremos los elementos que componen la interfaz de usuario de alto nivel. Esta interfaz usa componentes tales como botones, cajas de texto, formularios, etc. Estos elementos son implementados por cada dispositivo y la finalidad de usar las APIs de alto nivel es su portabilidad. Al usar estos elementos, perdemos el control del aspecto de nuestra aplicación ya que la estética de estos componentes depende exclusivamente del dispositivo donde se ejecute. En cambio, usando estas APIs de alto nivel ganaremos un alto grado de portabilidad de la misma aplicación entre distintos dispositivos. Fundamentalmente, se usan estas APIs cuando queremos construir aplicaciones de negocios.

2. Por otro lado tenemos las interfaces de usuario de bajo nivel. Al crear una aplicación usando las APIs de bajo nivel, tendremos un control total de lo que aparecerá por pantalla. Estas APIs nos darán un control completo sobre los recursos del dispositivo y podremos controlar eventos de bajo nivel como, por ejemplo, el rastreo de pulsaciones de teclas. Generalmente, estas APIs se utilizan para la creación de juegos donde el control sobre lo que aparece por pantalla y las acciones del usuario juegan un papel fundamental.

- Clase MIDlet

Un MIDlet es una aplicación realizada usando el perfil MIDP , como ya se ha comentado anteriormente. La aplicación debe extender a esta clase para que el AMS pueda gestionar sus estados y tener acceso a sus propiedades. El MIDlet puede por sí mismo realizar cambios de estado invocando a los métodos apropiados. Los métodos, entre otros, de los que dispone esta clase son los siguientes:

1. protected abstract void destroyApp(boolean incondicional)

   Indica la terminación del *MIDlet* y su paso al estado de "Destruido". En el estado de "Destruido" el *MIDlet* debe liberar todos los recursos y salvar cualquier dato en el almacenamiento persistente que deba ser guardado. Este método puede ser llamado desde los estados "Pausa" o "Activo".

2. public final void notifyDestroyed()

   Este método es utilizado por un MIDlet para indicar al AMS que ha entrado en el estado de "Destruido". En este caso, todos los recursos ocupados por el MIDlet deben ser liberados por éste de la misma forma que si se hubiera llamado al método MIDlet.destroyApp(). El AMS considerará que todos los recursos que ocupaba el MIDlet están libres para su uso.

3. public final void notifyPaused()

   Se notifica al AMS que el MIDlet no quiere estar "Activo" y que ha entrado en el estado de "Pausa". Este método sólo debe ser invocado cuándo el MIDlet esté en el estado "Activo". Una vez invocado este método, el MIDlet puede volver al estado "Activo" llamando al método MIDlet.startApp(), o ser destruido llamando al método MIDlet.destroyApp(). Si la aplicación es pausada por sí misma, es necesario llamar al método MIDlet.resumeRequest() para volver al estado "Activo".

4. protected abstract void pauseApp()

   Indica al MIDlet que entre en el estado de "Pausa". Este método sólo debe ser llamado cuándo el MIDlet esté en estado "Activo". Si ocurre una excepción RuntimeException durante la llamada a MIDlet.pauseApp(), el MIDlet será destruido inmediatamente. Se llamará a su método MIDlet.destroyApp() para liberar los recursos ocupados.

5. protected abstract void startApp()

Este método indica al MIDlet que ha entrado en el estado "Activo". Este método sólo puede ser invocado cuándo el MIDlet está en el estado de "Pausa". En el caso de que el MIDlet no pueda pasar al estado "Activo" en este momento pero si pueda hacerlo en un momento posterior, se lanzaría la excepción MIDletstateChangeException.

A través de los métodos anteriores se establece una comunicación entre el AMS y el MIDlet. Por un lado tenemos que los métodos startApp(), pauseApp() y destroyApp() los utiliza el AMS para comunicarse con el MIDlet, mientras que los métodos resumeRequest(), notifyPaused() y notifyDestroyed() los utiliza el MIDlet para comunicarse con el AMS.

- Clase Display

La clase Display representa el manejador de la pantalla y los dispositivos de entrada. Todo *MIDlet* debe poseer por lo menos un objeto Display, en el que podemos incluir tantos objetos Displayable como queramos. La clase Display puede obtener información sobre las características de la pantalla del dispositivo donde se ejecute el *MIDlet*, además de ser capaz de mostrar los objetos que componen nuestras interfaces.

Para obtenerla emplearemos el siguiente código:

```
Display pantalla = Display.getDisplay(this)
```

La llamada a este método la realizaremos dentro del constructor del *MIDlet*. De esta forma nos aseguramos que el objeto Display esté a nuestra disposición durante toda la ejecución de éste. Además, dentro del método startApp tendremos que hacer referencia a la pantalla que queramos que esté activa haciendo uso del método setCurrent(). Hay que tener en cuenta que cada vez que salimos del método pauseApp, entramos en el método startApp, por lo que la construcción de las pantallas y demás elementos que formarán parte de nuestro *MIDlet* la tendremos que hacer en el método constructor.

- Clase Displayable

La clase Displayable representa a las pantallas de nuestra aplicación. Como hemos dicho, cada objeto Display puede tener tantos objetos Displayable como quiera. Nuestras aplicaciones estarán formadas por varias pantallas que crearemos dentro del método constructor. Mediante los métodos getCurrent y setCurrent controlamos qué pantalla queremos que sea visible y accesible en cada momento.

- Clases Command y CommandListener

Un objeto de la clase Command mantiene información sobre un evento. Podemos pensar en él como un botón de Windows, por establecer una analogía. Generalmente, se implementa en nuestros *MIDlets* cuando queramos detectar y ejecutar una acción simple. Existen tres parámetros que hay que definir cuando construimos un objeto Command:

1. Etiqueta: La etiqueta es la cadena de texto que aparecerá en la pantalla del dispositivo que identificará a nuestro Command.

2. Tipo: Indica el tipo de objeto Command que queremos crear. La declaración del tipo sirve para que el dispositivo identifique el Command y le dé una apariencia específica acorde con el resto de aplicaciones existentes en el dispositivo.
3. Prioridad: Es posible asignar una prioridad específica a un objeto Command. Esto puede servirle al AMS para establecer un orden de aparición de los Command en pantalla. A mayor número, menor prioridad.

No basta sólo con crear un objeto Command de un determinado tipo para que realice la acción que se desea de acuerdo a su tipo. Para ello tenemos que implementar la interfaz CommandListener. Una interfaz es una clase donde todos sus métodos son declarados como abstract. Es misión nuestra implementar sus correspondientes métodos. En este caso, la interfaz CommandListener sólo incluye un método commandAction(Command c, Displayable d) donde se indicará la acción que queremos que se realice cuando se produzca un evento en el Command *c* que se encuentra en el objeto Displayable *d*.

- Clase Alert

El objeto Alert representa una pantalla de aviso. Normalmente se usa cuando queremos avisar al usuario de una situación especial como, por ejemplo, un error. Un Alert está formado por un título, texto e imágenes. Podemos elegir el tipo de alerta que vamos a mostrar. Cada tipo de alerta tiene asociado un sonido asociado. Los tipos que pueden ser definidos son los siguientes: ALARM, CONFIRMATION, ERROR, WARNING, INFO.

- Clase List

La clase List permite construir pantallas que poseen una lista de opciones. La clase List implementa la interfaz Choice y esto ofrece la posibilidad de crear 3 tipos distintos de listas cuyo tipo están definidos en esta interfaz y se muestran en la siguiente tabla:

Tipo	Descripción
EXCLUSIVE	Lista en la que un sólo elemento puede ser seleccionado a la vez.
IMPLICIT	Lista en la que la selección de un elemento provoca un evento.
MULTIPLE	Lista en la que cualquier número de elementos pueden ser seleccionados al mismo tiempo.

Tabla 3.8. Tipos de listas

A continuación se va a detallar cada una de estas listas:

1. Listas implícitas: En este tipo de listas, cuando se selecciona un elemento provocamos una acción. Éste mecanismo resulta muy útil para crear el clásico menú de opciones. La acción a ejecutar cuándo seleccionemos una opción tiene que ser implementada en el método commandAction(Command c, Displayable d).

2. Listas exclusivas: En este tipo de listas sólo se puede seleccionar un elemento a la vez. La selección de uno de ellos implica la deselección del resto. En este caso, al contrario que ocurre con las listas implícitas, la selección de una opción no provoca ningún tipo de evento, por lo que es necesario incluir un Command para salvar el estado del menú de opciones y

ver cuál de éstas ha sido seleccionada.

3. Listas múltiples: En estas listas podemos seleccionar los elementos que queramos al mismo tiempo. Como las listas exclusivas también es necesario incluir un Command para guardar el contexto.

- Clase TextBox

Una TextBox es una pantalla que nos permite editar texto en ella. Cuando se crea una TextBox, es necesario especificar su capacidad, es decir, el número de caracteres que queremos que albergue cómo máximo. Esta capacidad puede ser mayor que la que el dispositivo puede mostrar a la vez. En este caso, la implementación proporciona un mecanismo de *scroll* que permite visualizar todo el texto. También es posible añadir restricciones al texto que se puede incluir en una TextBox. Estas restricciones se encuentran en la clase TextField, íntimamenterelacionada con Textbox y algunas de ellas son: ANY, NUMERIc, PASSWORD, URL, etc.

- Clase TextField

Un TextField es un campo de texto que se puede insertar en un formulario y dónde es posible editar texto. Las diferencias entre las clases TextBox y TextField son:

1. Un TextField tiene que ser insertado en un formulario, mientras que un TextBox puede implementarse por sí mismo.

2. TextField deriva de la clase Item, mientras que TextBox deriva directamente de Screen, y sus eventos los controlamos a través de Commands. Por esta razón, los eventos que produce un TextField son controlados a través del método itemStateChanged(Item item), mientras que en un TextBox son controlados en el método commandAction(Command c, Displayable d).